引言：NoSQL

什么是NoSQL

Not Only SQL，不仅仅是SQL，泛指非关系型数据库

- 关系型数据库：操作数据时，使用sql（sql server、mysql等）
- 非关系型数据库：操作数据时，不是使用sql，有自己的语法

为什么会出现NoSQL

随着业务的发展，关系型数据库面对大规模和高并发的动态网站，出现了实现复杂、性能低的问题。

举例：

- 商城网站中对商品的频繁查询
- 热搜排行榜
- 订单超时问题
- 音频、视频存储

NoSQL四大分类

键值（Key-Value）存储数据库

- 说明：主要使用一个哈希表，表中由键值对组成（类比python中对字典）
- 特点：Key/Value模型优势在于简单、易部署
- 产品：Redis

列存储数据库

- 说明：用于分布式存储海量数据（十亿、百亿级数据）
- 特点：键仍然存在，但特点是指向了多个列
- 产品：HBase

文档型数据库

- 说明：以特定格式存储数据，比如：json。比键值数据库查询效率更高

```
# 数据举例
{id: 1, name: Zhangsan, age:10, score:100}
{id: 2, name: Lisi}
```

- 特点：以文档形式存储
- 产品：MongoDB

图形（Graph）数据库

- 说明：数据以灵活的图形模型进行存储
- 特点：没有标准的查询语言，通过接口进行查询
- 产品：Neo4J

NoSQL应用场景

- 数据模型简单
- IT系统对 数据灵活性 要求比较高
- 对数据库性能要求较高
- 不需要高度的数据一致性（比如：排行榜，数据错1～2个没关系）

Redis介绍

什么是Redis

> Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。

Redis 数据在内存中

- 优点：读写快
- 缺点：断电消失
- 解决断电消失的方案：持久化机制（内存数据会定期写入到硬盘中）

Redis特点

- 高性能key/value内存型数据库
- 支持丰富的数据模型（String、List、Set等）
- 支持持久化（内存数据会定期写入到硬盘中）
- 单进程、单线程 （线程安全，应用场景：分布式锁）

  “分布式锁” 应用场景说明：

  多个请求同时访问共享数据时，可能导致数据不同步

  =》加threading.Lock锁

  但当server部署在多台服务器上时，请求可能访问任意一台服务器上的server，但此时threading.Lock锁会只对其所在的server生效，无法解决多台服务器上的数据不同步的问题

  =》加分布式锁

Redis安装

Ubuntu安装Redis

```
apt update
apt install redis-server
```

Windows安装Redis

官方不提供Windows安装包，需要到github上获取安装包（比如：微软提供的安装包），本地解压即可使用

Redis使用前的准备

修改redis配置文件

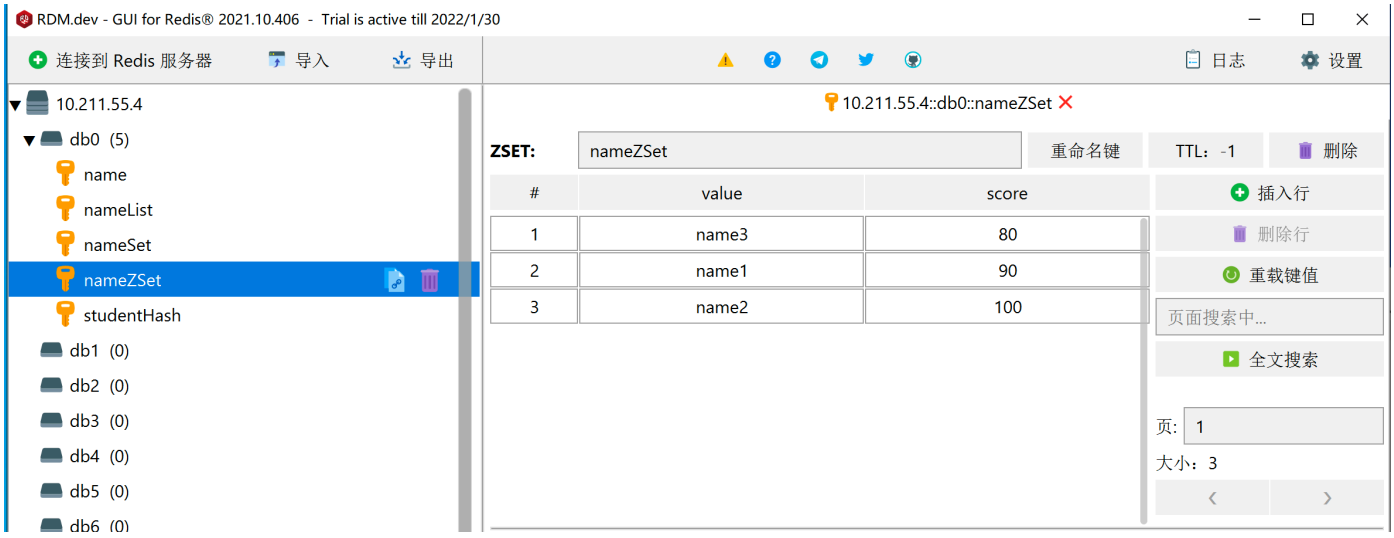
```
vim /etc/redis/redis.conf
```

需要修改的配置信息

```
# 允许来自任意一台PC上的客户端访问
bind 0.0.0.0
```

Windows可视化工具

Redis Desktop Manager



Redis基本操作

```
# 客户端连接redis服务端
redis-cli -h 10.211.55.4


# 选择库（默认16个库，编号：0～15，默认使用0号库）：select dbid(库编号)
# 注：库的个数可以通过修改配置文件进行调整。修改配置文件（/etc/redis/redis.conf）中的 databases 对应的值
```

```
select 0


# String类型
## 设置
set name Zhangsan
## 获取
get name


# List类型（有序 可重复）
## 设置
lpush nameList name1 name2 name3
## 获取
lrange nameList 0 -1


# Set类型（无序 不可重复）
## 设置
sadd nameSet name1 name2 name3 name2
## 获取
smembers nameSet


# ZSet类型（可排序的set集合，排序 不可重复）
## 设置
zadd nameZSet 85 name1 100 name2 80 name3 90 name1
## 获取
zrange nameSet 0 -1 withscores   # 默认升序


# Hash类型（value是 键值结构）
## 设置
hset studentHash name Zhangsan
## 获取
hget studentHash name


# 为key设置生存时间（单位：秒），当key过期时，会自动删除
expire name 10
```

Redis应用场景举例

- 具有时效性的功能
  - 验证码：超过60s失效
  - 未付款的订单：超过2h未付款，自动取消
  - token信息（安全令牌）
- 排行榜功能

- 分布式缓存

  将频繁被查询的信息，放到缓存中。请求优先从缓存中读取数据，如果没有再读数据库

- 分布式集群系统中，Session共享

  （Session：会话信息，存储客户端的身份信息，告诉web服务器当前请求是属于哪个会话的，用于区分用户）

- 分布式集群系统中，分布式锁

Redis持久化机制（2种）

什么是持久化

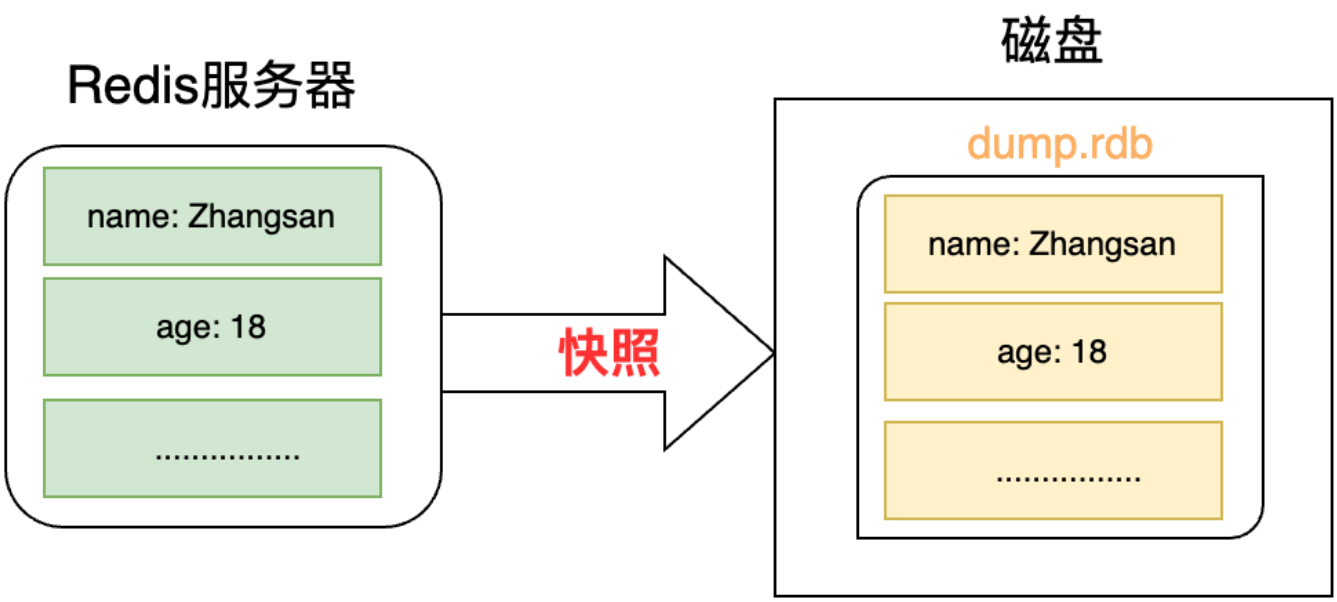
将内存中的数据写入到磁盘中

【持久化机制1】快照

特点

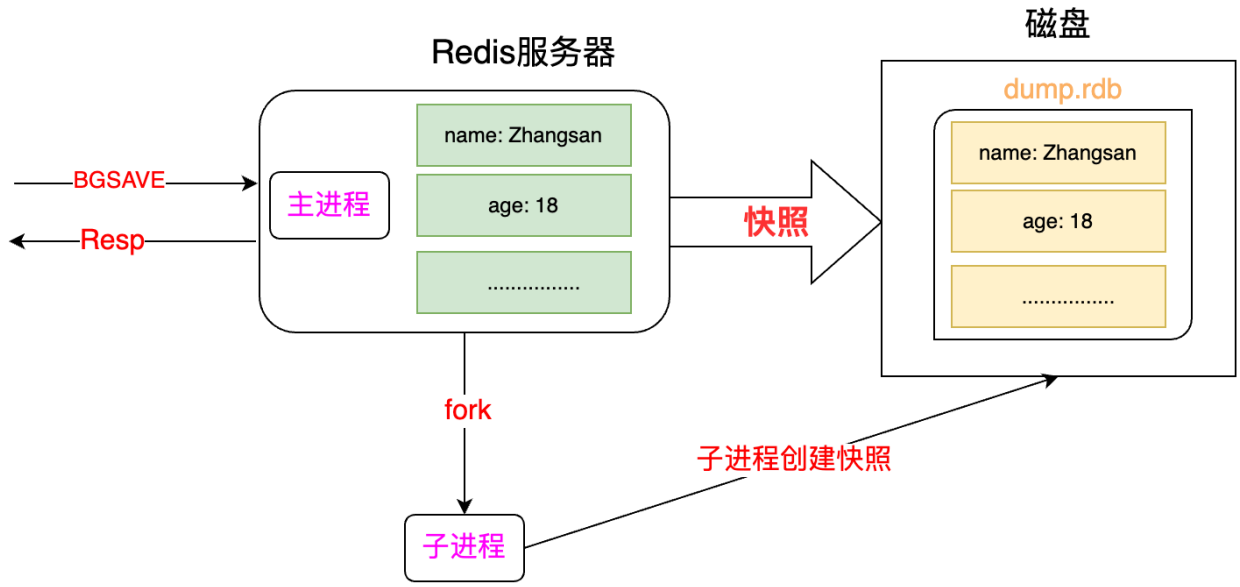
保存某一时刻的所有数据到磁盘（默认开启），默认保存到文件后缀.rdb，故也被称为RDB方式

（.rdb文件默认存储位置：/var/lib/redis/dump.rdb）

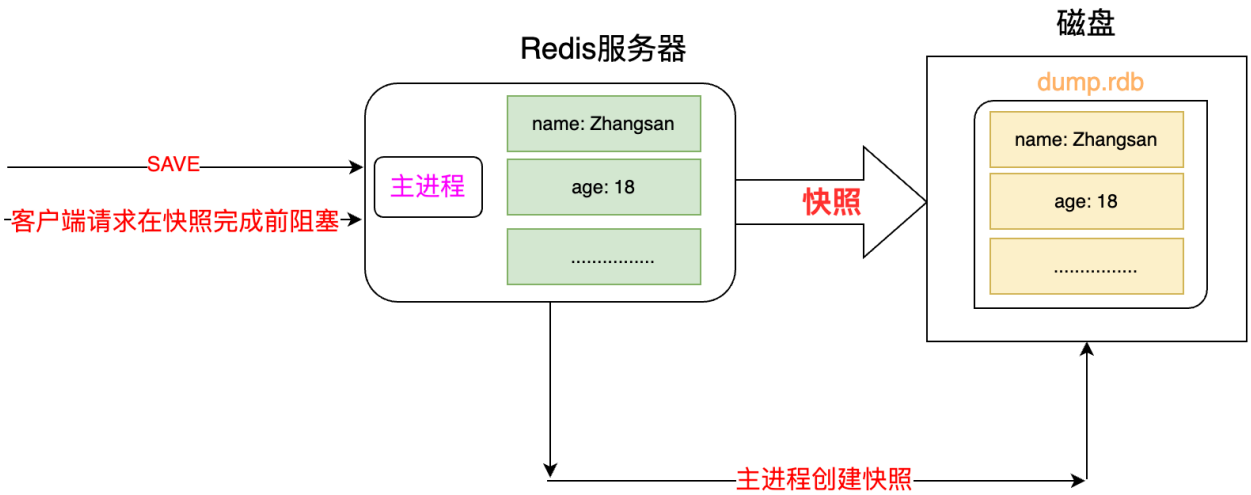


生成方式

- 客户端触发（2种方式）
  - BGSAVE：fork一个子进程创建快照，父进程继续处理命令请求



- SAVE（不常用）：主进程负责创建快照，创建过程中不能处理命令请求



- 服务器自动配置触发

  配置文件（/etc/redis/redis.conf）中 配置save的触发条件（触发BGSAVE）

缺点

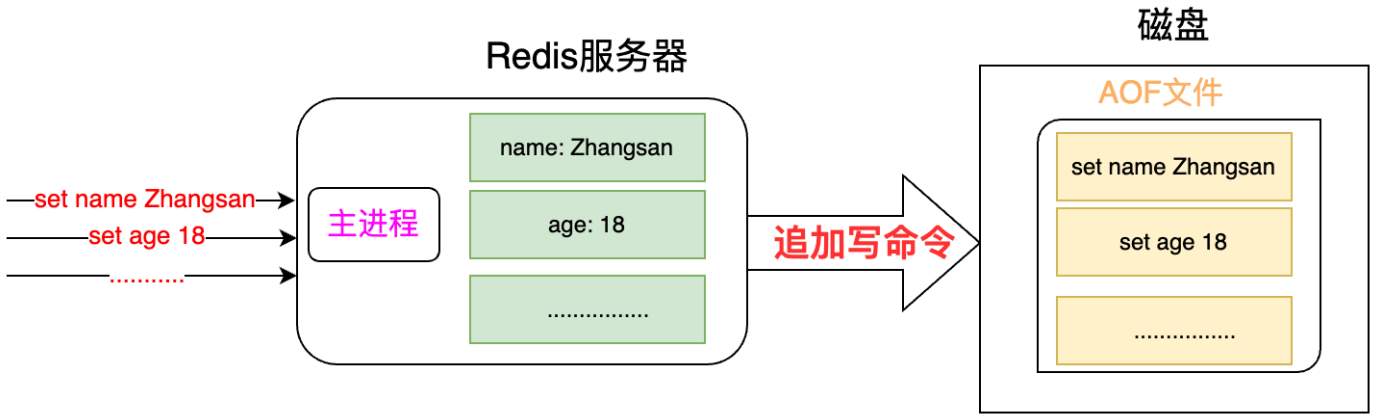
在t1时刻创建了快照，但是在t2时刻宕机，则 t1~t2 之间的数据丢失

【持久化机制2】AOF

特点

AOF（append only file），将所有客户端的写命令依次记录到AOF文件中。

当需要恢复数据时，redis会执行一次AOF文件所包含的所有写命令。



生成方式

在配置文件（/etc/redis/redis.conf）中，开启AOF持久化（默认没有开启）

```
# 配置文件中 “持久化” 选项
appendonly yes                          # 开启持久化
appendfilename "appendonly.aof"         # 指定持久化文件名称
appendfsync everysec                    # 指定追加频率
```

关于追加频率，有3种选项

- always（谨慎使用）
  - 说明：每个redis写命令都要同步写入磁盘。
  - 优点：在系统崩溃时，丢失的数据减少到最小
  - 缺点：由于频繁的对硬盘进行写入操作，会严重降低redis速度；对于固态硬盘，可能会引发 “写入放大” 问题，导致固态硬盘寿命由原来的几年降低为几个月
- everysec（推荐）
  - 说明：每秒执行一次同步，将多个命令写入磁盘。
  - 优点：兼顾了数据安全和写入性能，redis 每秒同步一次AOF文件时的性能 和 不使用任何持久化特性时的性能 相差无几
  - 缺点：在系统崩溃时，最多丢失1秒内产生的数据
- no（不推荐）
  - 说明：由操作系统决定何时同步
  - 优点：不会对redis性能产生影响
  - 缺点：系统崩溃时，丢失的数据不可控；当缓存了大量的写命令时，触发同步写入磁盘，redis处理命令速度会变慢

缺点

AOF文件占用大量磁盘空间

修改key为name的值10000次，会在AOF文件中记录10000条写入命令，导致AOF文件变得很大。

AOF重写

只记录最后一次修改key为name的值就好了，可以减少AOF文件的体积

触发重写的方式

- 客户端触发重写

  执行 BGREWRITEAOF 命令，不会阻塞redis服务

- 服务器配置自动触发

```
# 修改配置文件（/etc/redis/redis.conf）中的参数，触发自动重写
# eg：*64MB --> 20M --> *40MB --> 18MB --> *36MB
# 默认的配置：AOF文件体积越小，触发重写频率越高；体积越大，触发重写频率越低
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

原理

重写AOF文件的操作，并没有读取旧的文件，而是将整个内存中的数据库内容用命令的方式重写了一个新的AOF文件，再替换原有的AOF文件

总结

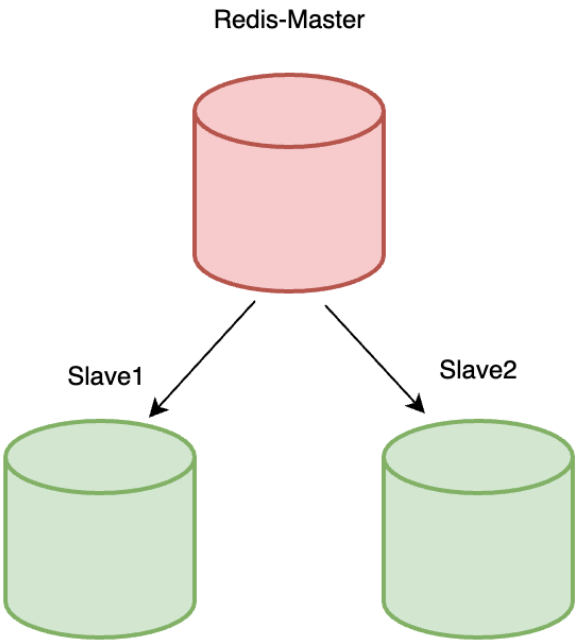
- 2种持久化可以同时使用（在redis启动时，会使用AOF文件恢复数据），也可以单独使用
- 除了将数据持久化到硬盘，建议在不同的地方备份

Redis主从复制架构

是什么

Redis数据的冗余备份

架构&&原理



Master节点数据变化时， Slave1、Slave2 触发数据同步。

（Slave节点只读，不允许写）

搭建方式

```
# 1、准备3台服务器并修改Redis配置
# Master 修改配置文件（/etc/redis/redis.conf）中的参数
port 6379
bind 0.0.0.0

# Slave1
port 6380
bind 0.0.0.0
replicaof MasterIp MasterPort

# Slave2
port 6381
bind 0.0.0.0
replicaof MasterIp MasterPort


# 2、启动3台服务器Redis服务
systemctl start redis-server.service
```

新的问题

主从复制架构只完成了备份。当Master节点挂掉时，Redis无法对外提供服务。

Redis哨兵机制

哨兵Sentinel机制

问题

当Master节点挂掉时，Redis无法对外提供服务。

目的

使Redis高可用

目标

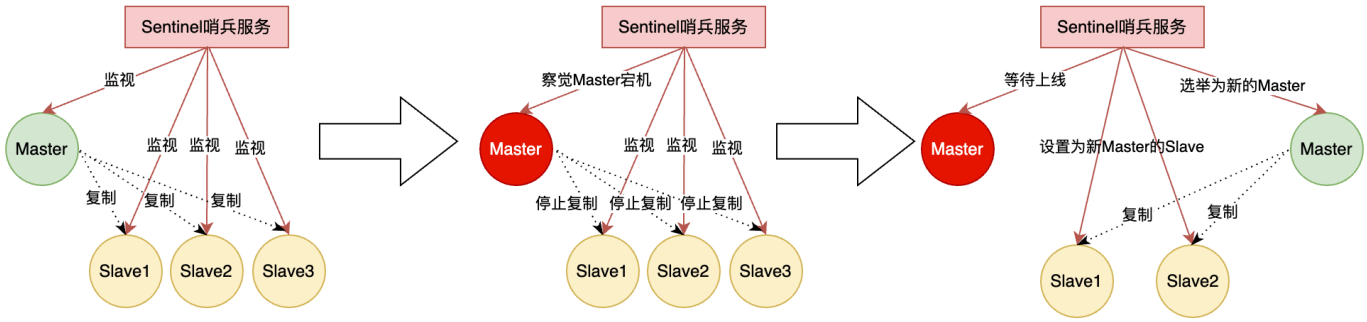
支持自动故障转移功能

解决方案

由一个或多个Sentinel监听任意多个Master服务器、以及这些Master服务器属下的所有Slave服务器。

当被监视的Master服务器进入下线状态时，自动将下线Master属下的某个Slave服务器升级为新的Master服务器。

架构&&原理



推荐使用多个Sentinel进行监听，原因：

若由于网络延迟，导致Sentinel误认为Master节点宕机，则会新推荐出一个Master，导致出现2个Master（脑裂）。

若存在多个Sentinel，会共同来判断原Master是否已宕机，以决定是否推荐出一个新的Master，降低脑裂发生的概率。

搭建方式

```
# 1、安装redis-sentinel
apt install redis-sentinel

# 2、配置哨兵，在sentinel.conf（/etc/redis/sentinel.conf）文件中填入内容
# ip：Master节点ip； port：Master节点port； 1：哨兵的个数
sentinel monitor 被监控的主从架构的名字（自定义） ip port 1

# 3、启动哨兵模式进行测试
redis-sentinel /etc/redis/sentinel.conf
```

新的问题

哨兵机制只完成了自动故障转移功能，还有2个问题无法解决：

单节点并发压力问题

当客户端并发访问量很大，一个Master节点无法支撑业务；

单节点内存和物理硬盘上限问题

由于Redis数据存储在内存中，如果数据量很大，会导致内存溢出；

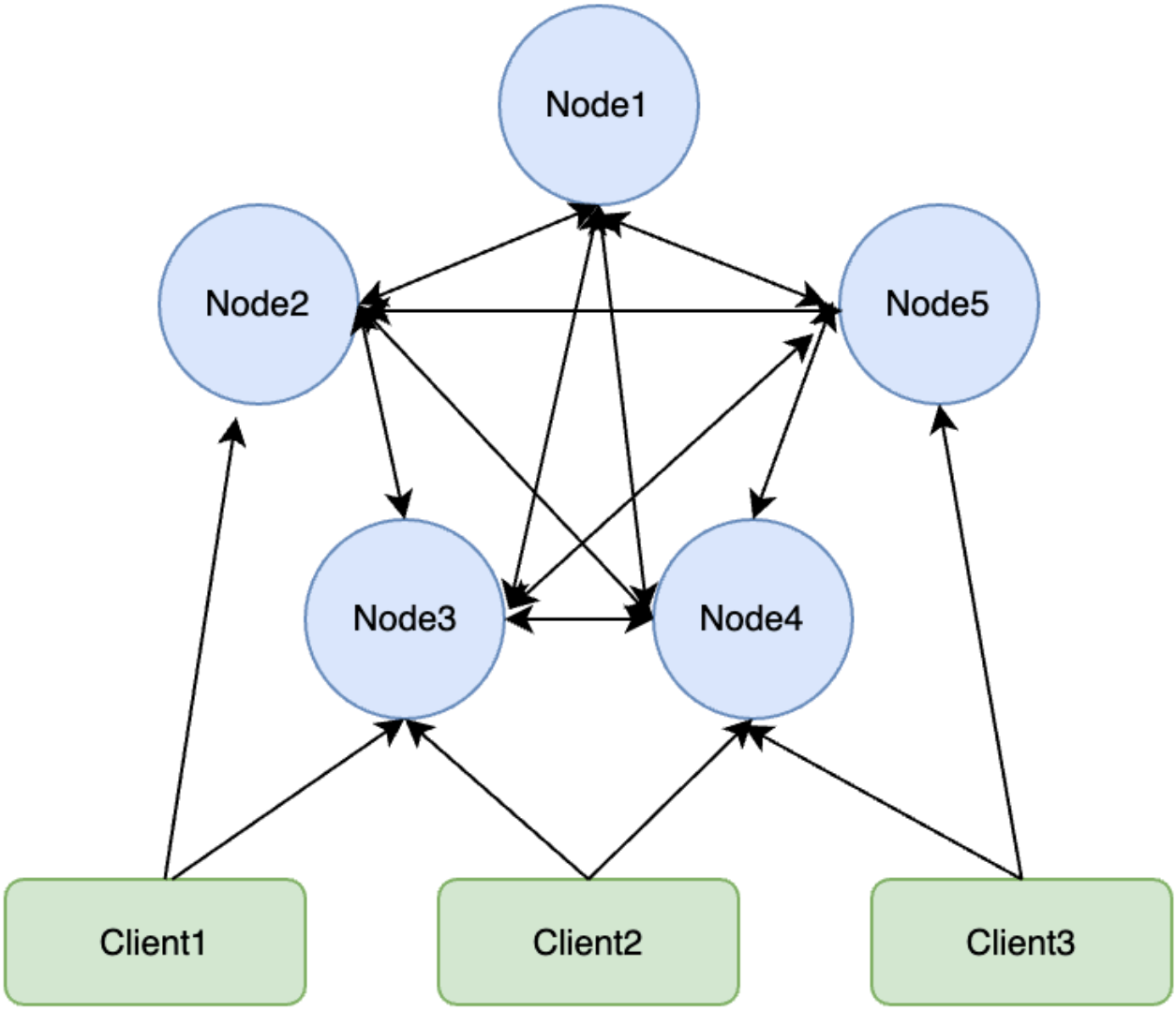
由于Redis持久化保存在物理磁盘中，如果数据量很大，也会导致物理磁盘被占满。

Redis集群

是什么

Redis在3.0后开始支持Cluster（集群）模式，支持节点的自动发现、Slave-Master选举和容错、在线分片等。

架构&&原理



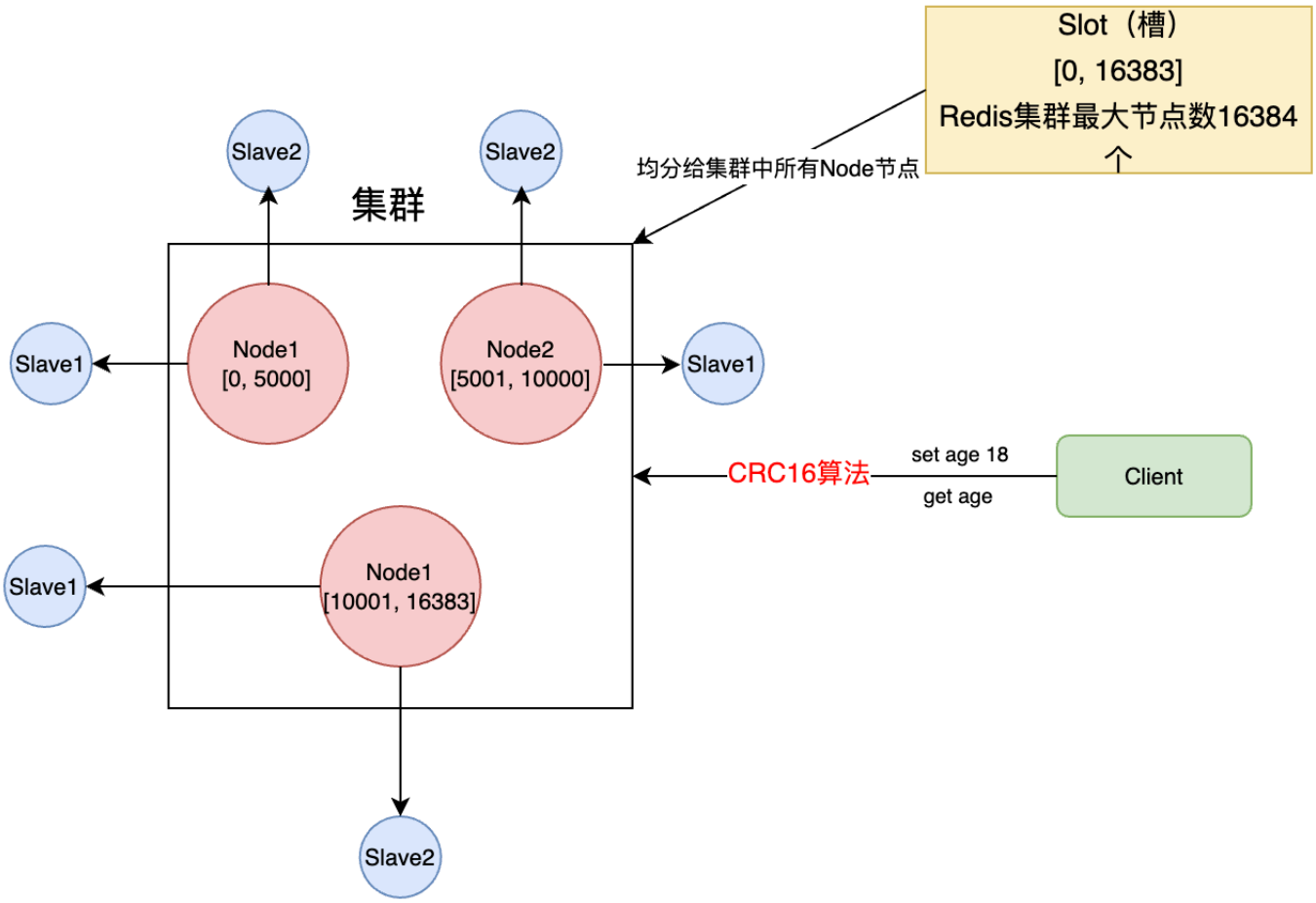
详细：

1、所有Redis节点彼此互联（PING-PIONG机制），内部使用二进制协议优化传输速度和带宽

2、节点的失效：集群中超过半数的节点检测失效时，才会判为失效的（因此节点数建议为奇数个）。

3、客户端与redis节点直连，不需要中间的Proxy层，客户端不需要连接集群中所有节点，连接集群中任意一个可用节点即可

4、redis-cluster把所有节点映射到 [0, 16383] slot上，集群负责维护 node <--> slot <--> value



CRC16算法

一、目的
Key通过CRC16算法找到Slot，将Vlaue存储到Slot中，这样就可以使Value存储在不同的Node中。

二、特点

1、集群模式下，所有key经过CRC16计算，计算结果范围在[0,16383]。==》避免超出Slot范围

2、同一个Key经过CRC16算法，计算结果一致。==》确保key找到唯一一个Slot

参考资料

- [1] Redis官网
- [2] Redis中文官网
- [3] Redis实战教程